SIMULATION-BASED LEARNING IN KNOWLEDGE-BASED CONTROLLERS

Phillip D. Stroud

LAUR-96-1987, Proc. 1996 IEEE Int'l Symposium on Intelligent Control, pp. 168-174, Sept 1996.

Los Alamos National Laboratory, Technology and Safety Assessment Division
Mail Stop F607, Los Alamos, NM 87545
stroud@lanl.gov

Abstract

A methodology is presented to transform knowledge-based controllers into adaptive structure representations. These adaptive controllers are then evolved using automatic directed search methods, where the performance of the trial controllers is evaluated through simulation in a synthetic environment. This automatically acquired knowledge is incorporated back into a revised knowledge-based controller. This process is developed and demonstrated using the flight and fire controllers of a simulated airborne laser system as a testbed.

Introduction

An important application for intelligent controllers is found in the world of simulation. In a synthetic environment built of interacting object-oriented actors, the use of intelligent controllers can transform the simulation into an entirely new tool. Simulation actors can be developed that emulate the behavior of human operators. This allows economic exploration of systems which would otherwise require the use of very expensive "human in the loop" simulation facilities. In addition, controllers can be constructed that learn, in an automatic way, from within a simulation environment. Controllers can "discover" new and better behaviors. They also have the capacity to adapt their behavior to new scenarios or changing conditions.

A controller is a part of an actor that decides which behavior to invoke, using data from its sensors. An adaptive controller maintains an internal representation that allows prediction of the effects of its various behaviors. It can change its input-output mapping based on its understanding of external conditions. An intelligent controller can generate new mappings, and form an evaluation of how good they are.

The approach, in essence, is to investigate how various kinds of rule or knowledge-based controllers can be represented by useful adaptive structures. A rule-based controller, or ruleset, maps the possible object and environment states into a set of object behaviors. The ruleset often consists of a set of IF-THEN-ELSE rules. Alternatively, the ruleset may perform this mapping by an algorithmic process.

An adaptive structure is an alternative representation of the state-behavior mapping to that provided by the ruleset. Three features characterize those adaptive structures which are useful for an intelligent controller. First, the structure must be capable of representing vast numbers of alternative rulesets, both similar and radically different from the original ruleset. Second, the adaptive structure must be easily trained to mimic a given original ruleset. Finally, the adaptive structure must be transparent, so that the meaning of any changes can be easily extracted and converted into a revised ruleset.

After conversion of a ruleset into an adaptive structure, a simulation environment is used to evolve the adaptive structure to find improved controllers. A second component of this investigation is an evaluation of various directed search methods. Because simulation based evaluation of the controller performance is relatively expensive, efficient search methods are essential.

The methodology was developed using an airborne laser (ABL) object in a theater ballistic missile defense simulation as a testbed. The airborne laser is a defensive system for destroying theater ballistic missiles during their boost phase. The airborne laser flight and fire controllers were used to demonstrate the process of transformation of a ruleset to an adaptive structure, the simulation based directed search for improved behavior, and the extraction of knowledge thus obtained.

Simulation Based Performance Evaluation

A simulation package ELASTIC (Evolutionary Los Alamos Simulation-based Training for Intelligent Controllers) was constructed to evaluate the performance of various controllers. The kernel of the simulation evaluates the number of missiles killed in a sortie. In a sortie, an airborne laser flies to its designated loiter area and begins surveillance. Theater missiles are launched at unknown times, from unknown launch locations, and to unknown targets. The launch zone, target zone, number and type of missiles are selected for consistency with a scenario of interest. ELASTIC generates a random missile launch script consistent with the scenario. The airborne laser is flown with a flight controller that can be either a script, a rule-based controller, or an adaptive structure. The missiles are flown with a 4 degree-of-freedom trajectory model, in order to obtain sufficient fidelity. ELASTIC maintains a track file of all the missiles currently in boost phase. A fire controller is used to select the next target, again with options for using rule-based or adaptive structure controllers. A high fidelity laser atmospheric propagation model is used to determine whether a lethal fluence is delivered to a missile before burn-out, again to provide adequate fidelity.

The laser power, wavelength, beam director aperture, pointing jitter and adaptive optics system were arbitrarily selected to provide partial coverage of the launch zone. The launch zone was taken to extend from 200km to 400km

ahead of the front of the ABL loiter box, with a width of 300 km. The loiter box width is 240 km. The plane maintains a constant altitude of 41kft, flies at Mach 0.85, and has a level flight minimum turning radius of 24.8 km. A nominal salvo consists of 110 missiles, with a spread in the missile range from about 400 to 600 km.

In order to obtain a statistically significant measure of performance for a given controller, many independent sorties must typically be simulated. For example, if the true mean kill probability was 50%, it requires the evaluation of 10,000 independent missile engagements to obtain an estimate with a 0.5% standard deviation in the error of the sample mean kill probability. ELASTIC generates and simulates enough sorties to give the desired statistical significance.

The Airborne Laser Flight Controller

The airborne laser flight control system has several conflicting objectives. First, it has to keep the plane within a certain rectangular region of air space. If the plane ventures through the front of this loiter box, the ABL is vulnerable to surface to air missiles. The sides of the box are set by the necessity to have flight corridors reserved for other air operations. The box width is typically 200 to 300 km. The second flight controller objective derives from the shape of the ABL field of fire. The field of fire of an ABL is cut off at about 115 degrees back from the nose of the plane, because of the physics of the air flow boundary layer around the turret, and the geometry of the plane. The ABL must fly to keep the field of fire over the launch zone to the greatest extent possible. Since the plane can turn at roughly two thirds of a degree per second without losing altitude, and there might be up to a minute available from the detection of a missile to the burnout of the missile, the ABL can greatly improve its performance by turning toward targets that are launched behind the field of fire. While turning to the target, the plane has to avoid leaving its assigned box. Another fire controller objective is to keep the plane as close to the threat area as possible without violating other constraints. The deliverable intensity drops off with range, so the kill probability per unit time is greater at closer range.

Knowledge-based ABL flight controller

In the past, two methods have been used to control the flight of an airborne laser platform within the context of a theater missile defense simulation. In three major computer simulations [1]-[3], the ABL is assigned a scripted orbit, such as a figure-eight, bow-tie or racetrack pattern. This approach is incapable of turning the plane towards a target, and then continuing in a sensible orbit. The ISSAC-ABEL simulation [2] allows the plane to yaw to the target, without leaving its scripted flight path. The EADSIM [3] simulation extends the field of fire as a substitute for turn to target capability. If properly implemented, these approaches might produce good overall performance estimates, but they introduce scenario dependent errors that can't be assessed from within their own context. An ABL node in the Theater Air Command and Control Simulation Facility (TACCSF) [4] has a human pilot manually flying the plane in a flight simulator. This real-time large scale simulation approach does not offer the ability to assess performance against tens or hundreds of thousands of trials.

The scripted orbit has a fundamental limitation: it can never respond to the threat. The ability to turn towards targets can greatly improve the ABL performance. There are other reasons, such as self-defense, that a platform would need to deviate from a scripted orbit. This limitation can be overcome by a flight controller that is able to issue sensible turn commands based on a requested turn, if any, and the plane heading and position relative to the box.

For constant altitude flight, the plane's state is described by three parameters: x and y, which give the location, and b, the relative bearing or heading. The coordinate system is defined relative to the loiter box, with the origin located at the center of the front of the box. The y direction is normal to the front of the box, toward the threat. The positive x direction is to the right when facing the threat. The relative bearing is the angle from the positive y direction to the direction of the plane's velocity, with positive defined to the right. In addition, the controller can receive a request for a turn. The fire controller in the simulation issues a request to turn toward a target if there is currently no target in the field of fire and there is at least one otherwise engageable target outside the field of fire. The request to turn is passed to the flight controller in a Boolean parameter, which is true if there is a request for a turn toward the threat, and false if there is no turn request. The controller input thus consists of two real variables, one real variable on the interval $-\pi$ to π, and one Boolean variable. For simplicity, the flight controller signal can be constrained to two possible values: 1 for a maximum forward turn and 0 for a maximum backward turn. Straight line flight is obtained by alternating control signals.

A flight control ruleset has been developed. The antecedent of each rule is evaluated in turn until one is found to be true. The baseline flight control ruleset is given by the following sequence:

front rule: IF (ahead of box) THEN turn backwards
front rule: ELSE IF (approaching box front from inside box) THEN turn backwards
side rule: ELSE IF (past or about to pass side) THEN turn forward if possible
side rule: ELSE IF (past or about to pass side) THEN turn backwards
retreat rule: ELSE IF (flying backwards) THEN turn forward
diagonal rule: ELSE IF (ahead of diagonal back to last resort circle) THEN turn backward
request rule: ELSE IF (received a turn request) THEN turn forward
setline rule: ELSE IF (ahead of baseline) THEN turn backwards
default rule: ELSE turn forward

This ruleset can produce a variety of bow-tie orbits, and is able to turn toward targets and then continue on sensible looking flight paths.

Transformation of the ruleset into an adaptive structure

Neural nets have been used to produce circular and figure-eight orbits [5] - [7]. The adaptive neural net controller approach suffers from the drawback that even if the net improves its performance through evolutionary training, it is difficult to translate the new and improved net weights into a new and improved ruleset. Methods for extracting rules from neural net weights require complex and computationally intensive searches for combinations of inputs that completely determine the net output [8], [9].

The ruleset can be cast into the form where the antecedents are conjunctions of simple Boolean atomic expressions. This is accomplished by defining a set of six abstracted Boolean variables {v_0, v_1, v_2, v_3, v_4, v_5} that characterize the flight controller inputs. The two additional variables that describe the plane's relation to the front of the box are treated as constraints. These six inputs are obtained directly from the ruleset antecedents:

v_0: Is the plane past or about to pass a side of the box?

v_1: Is the plane too close to the front to make a forward turn?

v_2: Is the plane headed forward of the max heading?

v_3: Is the plane ahead of the diagonal back to the last resort turning circle?

v_4: Is the plane in front of the setline?

v_5: Is there a request for a forward turn?

In terms of these Boolean variables, the ruleset output control signal can be expressed as

IF	(v_0==1 && v_1==1)	THEN c=0;
ELSE IF	(v_0==1 && v_1==0)	THEN c=1;
ELSE IF	(v_2 == 0)	THEN c=1;
ELSE IF	(v_3 == 1)	THEN c=0;
ELSE IF	(v_5==1)	THEN c=1;
ELSE IF	(v_4==1)	THEN c=0;
ELSE		c=1;

Since each of the six input variables can take two values, there are 2^6 =64 possible input states, and the ruleset assigns to each of them a control signal of either 0 or 1. The ruleset can thus be expanded into a completely equivalent set of 64 rules of the standard binary associative memory format [13]. The first rule, for example, would be

IF(v_0=0 AND v_1=0 AND v_2=0 AND v_3=0 AND v_4=0 AND v_5=0)
THEN c=1

Each of the 64 rules of the expanded equivalent ruleset can be represented as a column of the following matrix.

```
v0  0101010101010101010101010101010101010101010101010101010101010101
v1  0011001100110011001100110011001100110011001100110011001100110011
v2  0000111100001111000011110000111100001111000011110000111100001111
v3  0000000011111111000000001111111100000000111111110000000011111111
v4  0000000000000000111111111111111100000000000000001111111111111111
v5  0000000000000000000000000000000011111111111111111111111111111111
c   1110111011100100111001001110010011101110111001001110111011100100
```

The value of v_0 is in the first row, and so on. The bottom row shows the rule outputs. The original ruleset is completely encapsulated in the string of 64 binary digits that makes up the bottom row.

Conversely, each of the $2^{64} = 1.8 \times 10^{19}$ possible 64 bit strings represent alternative controllers. A simple reprioritizing of the rules within the ruleset, (such as honoring a turn request before avoiding crossing an edge of the box) would correspond to a different string of 64 bits. Most strings represent controllers that fly poorly, flying straight away from the box, in circles, or in strange, surprising trajectories. With so many alternative controllers, however, some of them are likely to perform better than the original ruleset. Also, some controllers would do better against particular classes of scenarios.

Evolutionary search for improved controllers

While any of these prospective controllers can be evaluated with the simulation, an exhaustive search would not be practical. Genetic algorithms [10] are ideally suited to search for alternative controllers that perform better than the original ruleset. The chromosome is the 64 bit string that represents a controller. The often difficult step of constructing a binary representation of the system [11] is already accomplished. A first generation population of controllers is constructed by generating a set of new 64 bit strings, each of which is a variation of the original ruleset string. Each variant is obtained by copying the ruleset controller string, but allowing a small probability that any bit will be flipped in the copy. Each new string will have a few bits different from the original string. The original ruleset is retained as a member of the original population. The performance of these new controllers is then evaluated with the simulation. The worst half of the original population of controllers are discarded. Parents of the next generation are selected from the remaining controllers, with selection probability depending on rank order. A new controller is formed from two parents by using single point cross-over: the descendent receives the first part of its bit string from one parent, and the last part from the other parent, with a random location of the cross-over point. The uniform cross-over method [12] was also tried, wherein the offspring has an even chance of inheriting any given bit from either parent. In this application, the two cross-over methods perform about the same. The new controllers are then mutated, with each bit having a small probability of flipping. The performance of the newly generated and mutated controllers is then evaluated, and they are sorted into the surviving controllers to make up the new generation. This process is iterated to obtain successive generations. The mutation rate is taken as an adaptive parameter.

Evolutionary training finds better controllers

The first attempt at evolutionary training used a training set of 25 sorties against random independent trickle launched salvos of 110 missiles each, for a total of 2750 simulated missile engagements. The baseline ruleset flew the plane in such a way that 1468 missiles were killed, for a single missile kill probability of 53.38%. The genetic search method was used to look for better controller strings, using the performance against the training set as the performance measure. Using a population of 18 strings, a string was obtained after 12 generations (requiring 117 performance evaluations) that was able to kill 1522 missiles, for an improvement over the ruleset of 54 additional missiles killed, and a kill probability increased to 55.3%. The chance of obtaining this result if there had been no real improvement in the controller is found via the student's-t test to be 7%, so there is a 93% confidence that the controller improved. Noting that approximately 900 of the missiles are out of range regardless of the flight path, the observed improvement is quite impressive.

The evolved controller string can be used to directly modify the baseline ruleset by adding three new rules at the beginning of the ruleset:

IF(v_0=0 AND v_1=0 AND v_2=1 AND v_3=1 AND v_5=0)
THEN c=1

IF(v_0=1 AND v_1=0 AND v_2=0 AND v_3=0 AND v_4=0 AND v_5=0)
THEN c=0

IF(v_0=0 AND v_1=0 AND v_2=0 AND v_3=0 AND v_4=1 AND v_5=0)
THEN c=0

The first of these, combining bits 12 and 28, issues a forward turn command when the plane 0) is not past or about to pass a side of the box, 1) has room to make a forward turn, 2) is not headed too far backwards, 3) has room to get to the diagonal back to the last resort turning circle, and 5) had no request for a turn. The controller has learned that under these circumstances, the plane should not fly back to the setline, as it would have done under the baseline ruleset. The other new rules incorporate the new knowledge that under certain circumstances, it is better to make a full rearward turn.

The controller evolved on the training set was then tested against 100 independent random salvos of 110 missiles. The baseline ruleset killed 5925 of the 11000 missiles in this test set. The evolved controller killed 5990 of them. The controller that was evolved on one set of salvos was able to perform better than the ruleset on a completely independent set of salvos, improving the kill probability from 53.9% to 54.5%. The chance that this result would be obtained with no actual improvement is found to be 19% via the student's T test, giving an 81% confidence that the controller did improve. Some of the adaptation that the controller made to the training set is seen to be specific to that training set, while some of the adaptation generalizes to the new test set.

Another attempt to evolve a controller produced similar results. This time, a set of 50 salvos of 110 missiles each was used to evaluate the performance in the evolutionary training process. The ruleset controller killed 2960 of 5500, for a kill probability of 53.8%. For this round of training, there were 24 strings in the population. After 72 evaluations (requiring about a half hour each), a string had been obtained that killed 3009 of 5500 missiles, for a kill probability of 54.7%. When this new controller was evaluated against an independent set of 100 salvos, it was able to kill 5903 of 11000 missiles, where the baseline ruleset controller got 5844. Again, about half of the improvement obtained by evolving a controller against a training set of salvoes is retained when the evolved controller is used against an independent testing set of salvos.

When the performance against a single scripted salvo is used to evolve a controller, the performance can be spectacularly enhanced, allowing as many as 8 extra kills out of 110 engagements. However, the resulting controllers are very specifically adapted to the training set, and on average perform worse than the baseline ruleset when evaluated against independent test sets. This result points out the danger of developing tactics or rulesets based on training against only a few scripted series of launches, repeated over and over.

The Airborne Laser Fire Controller

The ABL fire controller has the function of selecting the best target from a queue of boosting missiles in a track file. The baseline rule-based fire controller uses an algorithm to estimate which target in the queue could be destroyed in the shortest amount of time. The time-to-kill estimate is the sum of the slew time (to rotate the turret to the direction of the target) and the dwell time (equal to the lethal fluence divided by the deliverable intensity). The deliverable intensity is a complicated function of a number of parameters: the range to target, the target altitude, the azimuth angle from the nose of the plane to the target, the target aspect angle, target and platform velocities, atmospheric conditions, etc. The fire controller output is a pointer to the target with the shortest estimated time-to-kill. This simple controller was able to perform at about the same level as a human operator manually selecting targets [1].

There are several obvious avenues for prospective improvement of this baseline algorithm. The first employs an estimate of the probability that a missile won't burn out prior to receiving a lethal fluence. The target selection is then based on maximizing the kill probability per unit time. A second avenue is chess-like. All targets in the queue are evaluated for estimated time-to-kill or kill probability per unit time. The best three or four are then evaluated at higher fidelity, looking at all possible permutations in firing order. The avenue of interest here, however, is to transform the baseline algorithmic controller into an adaptive structure, and then let the adaptive controller learn better behavior automatically.

The first step in developing a useful adaptive representation of the algorithmic controller is to identify the important state parameters. Three parameters were found to dominate: the range to target, the target elevation, and the slew angle. The adaptive structure is like a black box that takes these three inputs and produces a priority

value for each target in the queue. The black box has some adjustable knobs that vary the mapping from input to output. It differs from a black box, however, in that it is transparent: the knob parameters have meaning.

The extended multi-linear expansion

A very simple adaptive structure configuration, the multi-linear expansion, was selected for this evaluation. A linear transformation of one input x to output y can be written $y = w_1(1-x) + w_2 x$. This is the equation of a line with y-intercept of w_1 and slope of w_2-w_1. It is also a linear combination of two functions of x, namely h_1=1-x and h_2=x. The expansion coefficients have meaning: w_1 is the output when the input is low (x=0), and w_2 is the output when input is high (x=1). This linear transform can be extended to characterize more than two input states. For example, if $\{w_1,w_2,w_3\}$ represent the output for low, medium and high input values, the expansion can be written $y = w_1 T(x-0) + w_2 T(x-0.5) + w_3 T(x-1)$, where T(x) is a function that has a value of 1 for an argument of 0, and falls off to zero as |x| gets large. If T(x) is symmetric, it can be called a radial basis function. If T(x) is a triangular function (i.e. T(x)=Max(0,1-|x/b|) where b is the triangle half-base) this expansion has a direct fuzzy logic interpretation. The basis function can take other shapes, such as Gaussian, top-hat, bi-triangular, or trapezoidal. The transform can be seen as a linear combination of localized basis functions, each centered at some input value. Any functional mapping (except one with an infinite number of discontinuities) can be represented by this type of expansion, as long as enough centers are used.

When there are more than one input parameter, the linear expansion can be replaced with a multi-linear expansion. In the experNet configuration [13], the expansion functions are the 2^D multi-linear combinations of a set of D input parameters. For the case of 3 input dimensions, where the input set is $\{x_1, x_2, x_3\}$, the 8 expansion functions are

$h_1 = (1-x_1)\ (1-x_2)\ (1-x_3)$
$h_2 = x_1\ (1-x_2)\ (1-x_3)$
$h_3 = (1-x_1)\ x_2\ (1-x_3)$
$h_4 = x_1\ x_2\ (1-x_3)$
$h_5 = (1-x_1)\ (1-x_2)\ x_3$
$h_6 = x_1\ (1-x_2)\ x_3$
$h_7 = (1-x_1)\ x_2\ x_3$
$h_8 = x_1\ x_2\ x_3$

The x_k can represent binary variables, in which case the h_j are also binary valued functions specifying whether or not the input state matches a particular one of the 2^D possible states. The x_k can also represent continuous real values in the interval [0,1]. This can be interpreted as a fuzzy logic version of the binary variable case, where the x_k and h_j are interpreted as membership functions [14]. The above expansion can be extended to more than two expansion functions in each input dimension. An expansion of each of three dimensions into "low", "medium", and "high" fuzzy values would combine to give 27 expansion functions. Each expansion function can be interpreted as a basis function centered at some input state. The "net" output is a linear combination of these functions of the input vector, with the coefficient of the j^{th} function being called w_j. The net output is

$$y = \sum_{j=1}^{H} w_j h_j$$

For the adaptive ABL fire controller, x_1 represents the ground range from the platform to the target. $x_1 = 0$ corresponds to a ground range of 100 km, while $x_1 = 1$ corresponds to a ground range of 700 km. The second state parameter, x_2, represents the target altitude above the ground ($x_2 = 0$ corresponds to a TBM altitude of 5 km while $x_2 = 1$ corresponds to a TBM altitude of 75 km). The third, x_3, is the normalized angle from the current beam turret bearing to the target bearing. For $x_3 = 0$, the turret is pointed directly at the target, while for $x_3 = 1$, the turret would have to slew through 120 degrees to point at the target.

Supervised training

After the adaptive structure is constructed, the set of weight values must be found that lets the multi-linear net output best mimic the original ruleset. This process is known as supervised training. It does not require simulation based performance evaluation. The standard procedure is to generate a training set of input-output pairs using the original controller. A search is then performed for the set of weights that minimizes the sum, over all pairs, of the squared difference between the net output and the training set output value. In this form, supervised training is equivalent to least squares regression.

A baseline training set of 1000 vector - output pairs was generated, where each vector contains three parameters selected from a uniform distribution in (0,1), and each output is the corresponding kill rate. A little multi-linear net was constructed, with the 8 expansion functions given above. In addition, a larger multi-linear net was constructed with 27 expansion functions, using low, medium and high centers in each of three dimensions. A first approximation to the set of weights is given by the ruleset values obtained at the corresponding expansion function centers.

The widely implemented neural net training method, back propagation of error derivatives, is based on a gradient descent optimization strategy. The weight vector undergoes a process of iterative improvement. One of the training vectors, say the i^{th} one, is selected. The weight vector is incremented in the direction opposite of the gradient of the

error in weight space. The size of the increment is taken as some fraction (known as the learning rate) of the distance to the minima that would obtain for a quadratic error surface. By repeated cycling through the training set, the weight vector might converge to a solution that minimizes the mean square error over the whole training set.

Using an adaptive learning rate, gradient descent with individually presented training vectors obtains a solution with rms error within 1% of the optimal solution in five complete cycles through the training set. This method requires 5000 performance and gradient evaluations whether H is 8 or 27, and whether the weights are initialized to 0 or to the first approximation described above.

Because of the simple structure of the multi-linear expansion, more efficient gradient descent and matrix inversion methods can be used. A much more efficient gradient descent scheme is based on presentation of the whole training set at once. This batch-mode gradient descent obtains a solution with rms error within 1% of the optimal solution with 20 performance and gradient evaluations if the weights are initialized to zero, and with 8 performance and gradient evaluations when the weights are initialized to the approximation described above (for both the 8 and the 27 function expansion). Inversion of the matrix of the normal equations, singular value decomposition of design matrix, and singular value decomposition of matrix of the normal equations were all found to be much more efficient than gradient descent methods.

Production training

The supervised training methods described above are not applicable to simulation-based learning. The performance surface (i.e. the simulation-based performance as a function of the controller input variables) will have many local optima, rendering gradient descent search strategies useless.

The Pivot and Random Offset method starts with a set of weights (the pivot), and evaluates the performance of the net (or the sum of square errors relative to the original ruleset for the case of supervised training). Then an offset is obtained by applying a random increment to the pivot weight vector, and the performance of the offset is evaluated. If the performance is improved, the offset weight vector is accepted as the new pivot. Otherwise, a new offset is tried. If the pivot is replaced by an offset, a new offset is attempted in the same direction as the previous one. A simulated annealing approach has been implemented to allow escape from local minima. The offset is always accepted if its performance exceeds that of the pivot. With simulated annealing, the offset is also occasionally accepted when its performance is worse. The probability of acceptance of a worse solution is given by a Boltzmann distribution at "temperature" T, which has the same units as the measure of performance. The temperature is gradually reduced (using a power law annealing schedule) until there is no likelihood of accepting a worse offset.

A rough characterization of the efficiency of the pivot-offset strategy can be obtained by applying it to the supervised training task. The performance of this pivot-offset method is stochastic, depending sensitively on the starting random seed. For the 27 expansion function case with 1000 training vectors, starting with weights initialized to 0, a series of 13 runs required as few as 1844 performance evaluations, or as many as 3806 to converge to a solution with rms error within 1% of the error of the optimal solution. The mean requirement is 2588 evaluations. This compares with the 20 evaluations required to reach this accuracy with the batch decent method, although no gradients have to be evaluated.

If the initial set of weights is set equal to the true values obtained at the corresponding expansion function centers, the required number of performance evaluations ranges from 967 to 2856, with a mean (in 13 runs) of 1985. This gives some insight into the improvement in production training that might be obtained by starting in the neighborhood of a good answer. For the H=8 cases, the number of required performance evaluations is 221 when starting from initial weights of 0, and 143 when starting from weights initialized to the target values at the expansion function centers. Note that with traditional multi-layer perceptron neural net configurations, there is no way to initialize the weights to match a good solution.

Fire controller proof-of-principle

For evaluation of the performance of fire controllers, the baseline scenario geometry as described above is used, with a salvo consisting of 100 launches in a three minute period. For this scenario, the shortest time-to-kill fire controller destroyed 69 of 100 missiles. A simple eight-parameter multi-linear expansion was trained to mimic the shortest time-to-kill fire controller. It was able to destroy 68 out of the same 100 missiles. After the multi-linear expansion weights were allowed to evolve, using a pivot-offset directed search with simulation-based performance evaluation, a new controller was obtained that destroyed 76 of the 100 missiles. Furthermore, this new set of weights has direct interpretation as a revised target prioritizing ruleset.

Conclusion

An approach has been developed for implementing intelligent controllers into simulation environments. The essence of the approach is to transform from an existing rule or knowledge-based controller into a useful adaptive structure. The adaptive structure must be capable of representing the behavior of the original ruleset, as well as a vast number of similar and radically different behaviors. The adaptive controller is first trained to mimic the original knowledge-based controller, using well characterized supervised training methods. Directed search strategies are then used to evolve the adaptive controller towards improved performance, where the performance of

trial controllers is evaluated from within the simulation environment. Finally, the improved controller is transformed back to a revised ruleset so that people can understand what it has learned.

Several directed search methods have been investigated for this purpose. In the flight controller example, where controller mapping is represented by a binary associative map of high dimension, the methods of genetic programming were found to be suitable and efficient. In the fire controller example, the pivot-offset method with simulated annealing was able to find improved controllers. In both of these very different cases, automatic rule discovery was able to produce significant improvements.

The problem space (the set of all possible input states combined with the set of all attainable rule set mappings) for these two testbed controllers was sufficiently large to demonstrate the validity of the methodology. Much more dramatic improvements in performance can be obtained by using this methodology with more elaborate rule sets and state representations. For example, the width of the loiter box is fixed in the testbed flight control rule set. A trivial extension of the state representation would allow the controller to adapt the location of the box sides to best cover a particular threat of interest. Likewise, a simple extension of state representation to include pairs and triplets of missiles in the track file would greatly extend the problem space of the fire controller, and would with a practical certainty include better fire controllers than the simple testbed was able to represent.

Nothing in this process is specific to fire or flight controllers, except the representation of the state and the rule set mapping. This methodology of transforming a knowledge-based controller to an adaptive structure, evolving the weights automatically in the context of a synthetic environment, and transforming back to an improved ruleset, applies to a wide variety of systems. Even a small automatically produced improvement in a major expert system application could have a large pay-off, financial or otherwise.

The methodology described in this paper provides for automatic off-line simulation-based learning of knowledge-based controllers. This methodology points to the next step in the progression of intelligent controllers: real time or on-line learning capability. We are now investigating the extension of this methodology to achieve on-line learning capability by incorporating the whole evolutionary machinery and populations of behaviors and internal simulations within a controller.

References

[1] TEMPEST (TACCSF Exploratory Model of Performance, Strategy, and Tactics), numerical simulation package copyright by S. Mortenson, Los Alamos National Laboratory and the Regents of the University of California, 1992.

[2] ISSAC-ABEL, a proprietary simulation package developed by W.J.Shaeffer and Associates.

[3] The Extended Air Defense Simulation (EADSIM) User's Reference Manual, US Army SSDC, Huntsville Alabama, 1993.

[4] J. Brown, C. Heydemann, J. Soukup, "Theater Air Command and Control Simulation Facility ABL Test 6," Airborne Laser Program report, Phillips Laboratory, Albuquerque NM, 1994.

[5] D. Lin, J. Dayhoff, and P. Ligomenides, Trajectory Production With the Adaptive Time-Delay Neural Network," Neural Networks, vol. 8, pp. 447-461, 1995.

[6] N. Toomarian, "Learning a trajectory using Adjoint Functions and Teacher Forcing," Neural Networks, vol. 5, pp. 473-484, 1992.

[7] B. Pearlmutter, "Learning State Space Trajectories in Recurrent Neural Networks," International Joint Conf. on Neural Networks, Washington, II, pp. 365-372, 1989.

[8] L. Fu, "Rule Generation from Neural Networks," IEEE trans. on Systems, Man, and Cybernetics, vol. 24, pp. 1114-1124, 1994.

[9] M. Craven, and J. Shavlik, "Using Sampling and Queries to Extract Rules from Trained Neural Networks," Machine Learning: Proc. of the Eleventh International Conf., W.W.Cohen & H.Hirsh, eds., Morgan Kaufmann, San Francisco, 1994.

[10] D.E.Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Publishing Co, 1989.

[11] J.R.Oliver, "Discovering Individual Decision Rules: an Application of Genetic Algorithms," Proc.5th International Conf on Genetic Algorithms, S.Forrest ed., Morgan-Kaufman Publishers, San Mateo, CA, Jul 1993.

[12] L.J.Eshelman, J.D Schaffer, "Crossover's Niche," Proc.5th International Conf on Genetic Algorithms, S.Forrest ed., Morgan-Kaufman Publishers, San Mateo, CA, Jul 1993.

[13] C.Barrett, R.Jones, U.Hand, "Adaptive Capture of Expert Knowledge", Los Alamos National Laboratory Technical Report LA-UR-95-1391, 1995.

[14] B.Kosko, Neural Networks and Fuzzy Systems, Prentice-Hall, 1992.